SIT232 Object Oriented Development
Task 7.2C

By Venujan Malaiyandi
BSCP|CS|61|101

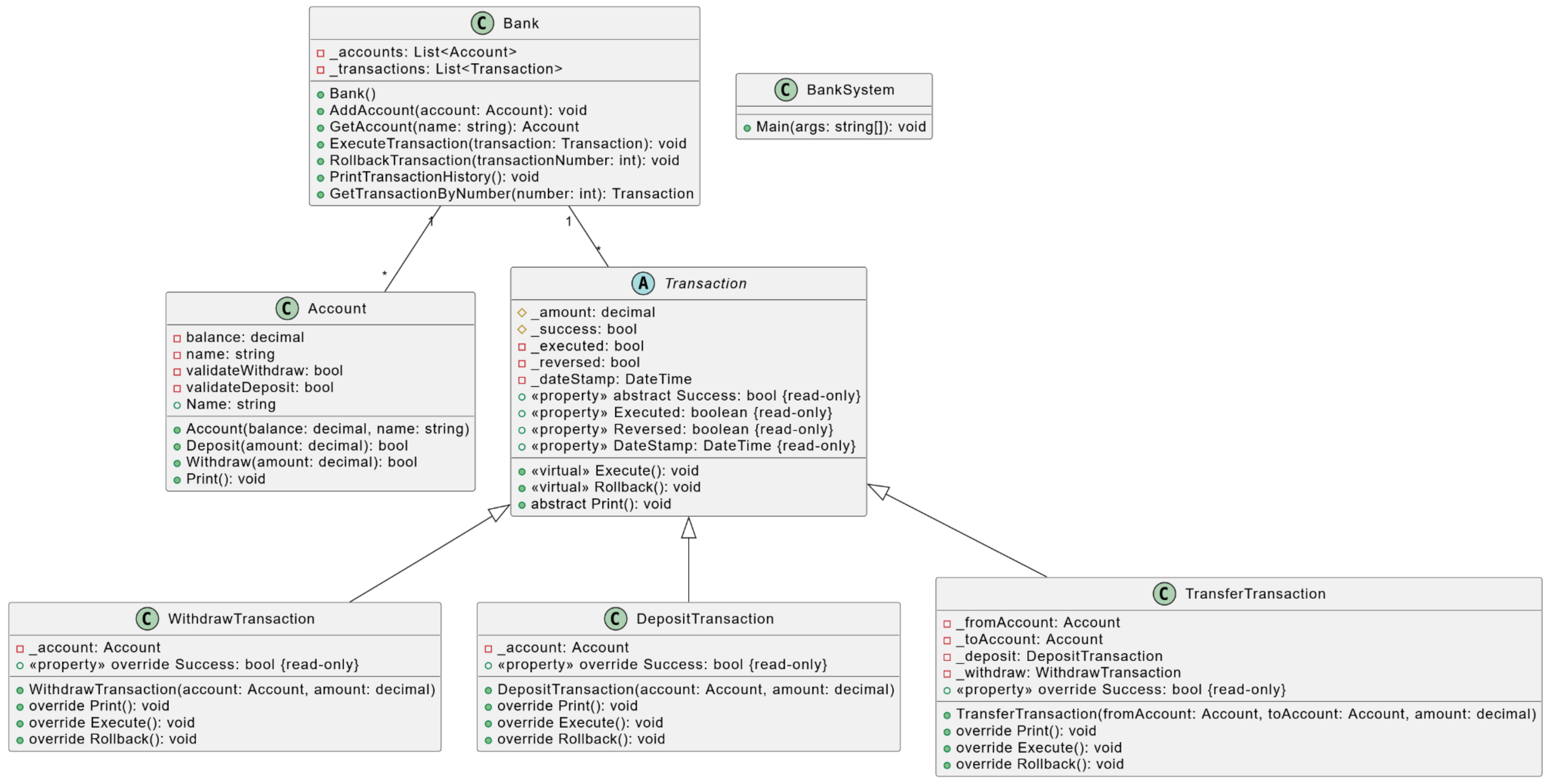
1. How do you represent classes, fields, methods, and relationships in terms of the Universal Modelling Language?
   In the Universal Modeling Language (UML), classes, fields, methods, and relationships are represented as follows:
   a. The classes are represented as a rectangle with three compartments:
      i. The top compartment contains the class name.
      ii. The middle compartment contains the class attributes (fields).
      iii. The bottom compartment contains the class methods.

   b. Fields are represented in the middle compartment of the class rectangle. They are written with the access modifier (+ for public, - for private, # for protected) followed by the field name and its type.

   c. Methods are represented in the bottom compartment of the class rectangle. They are written with the access modifier (+ for public, - for private, # for protected) followed by the method name, parameter list (if any), and return type.

   d. Relationships between classes are represented by lines connecting them. Common types of relationships include:
      i. Association: A connection between two classes, indicating that one class is associated with another. It's represented by a solid line to the associated class.
      ii. Inheritance/Generalization: Indicates that one class is a subclass (or derived class) of another. It's represented by a solid line with an arrow pointing to the superclass.
      iii. Composition: Indicates that one class owns or contains another class. It's represented by a filled diamond-shaped arrow pointing to the owned class.
      iv. Aggregation: Similar to composition but with a weaker relationship. It indicates that one class may contain another, but the contained class can exist independently. It's represented by an empty diamond-shaped arrow pointing to the owned class.

2. Describe the relationship between the diagram and your code.
   The diagram for the created Banking System is given below. **Additionally, a much more visible version of the diagram is attached inside the Task folder, as a Venujan-SIT232_Task7.2C-corrected-uml-diagram.svg.**


Bank
_accounts: List<Account>
_transactions: List<Transaction>
Bank()
AddAccount(account: Account): void
GetAccount(name: string): Account
ExecuteTransaction(transaction: Transaction): void
RollbackTransaction(transactionNumber: int): void
PrintTransactionHistory(): void
GetTransactionByNumber(number: int): Transaction
BankSystem
Main(args: string[]): void
1
*
1
*
Account
balance: decimal
name: string
validateWithdraw: bool
validateDeposit: bool
Name: string
Account(balance: decimal, name: string)
Deposit(amount: decimal): bool
Withdraw(amount: decimal): bool
Print(): void
Transaction
_amount: decimal
_success: bool
_executed: bool
_reversed: bool
_dateStamp: DateTime
«property» abstract Success: bool {read-only}
«property» Executed: boolean {read-only}
«property» Reversed: boolean {read-only}
«property» DateStamp: DateTime {read-only}
«virtual» Execute(): void
«virtual» Rollback(): void
abstract Print(): void
WithdrawTransaction
_account: Account
«property» override Success: bool {read-only}
WithdrawTransaction(account: Account, amount: decimal)
override Print(): void
override Execute(): void
override Rollback(): void
DepositTransaction
_account: Account
«property» override Success: bool {read-only}
DepositTransaction(account: Account, amount: decimal)
override Print(): void
override Execute(): void
override Rollback(): void
TransferTransaction
_fromAccount: Account
_toAccount: Account
_deposit: DepositTransaction
_withdraw: WithdrawTransaction
«property» override Success: bool {read-only}
TransferTransaction(fromAccount: Account, toAccount: Account, amount: decimal)
override Print(): void
override Execute(): void
override Rollback(): void

The above attached diagram paints an accurate representation of relationships between the classes from my code. **WithdrawTransaction**, **DepositTransaction**, **TransferTransaction** are child classes, inherited from the abstract class **Transaction**.
There is a one-to-many association between **Bank** class and the **Account** class. This means one **Bank** can have many **Account**s.
There is also one-to-many association **Bank** and the **Transaction** class.

3. How could you use this to think through a solution before you write the code? What would be the advantage of doing this?
    a. **Visualizing Relationships**: UML diagrams provide a visual representation of how classes are connected and interact with each other. This helps in understanding the overall structure of the system.

    b. **Identifying Class Responsibilities:** It helps in clarifying the responsibilities and roles of each class. This ensures that each class has a clear purpose and helps maintain a clean and organized codebase.

    c. **Planning Class Attributes and Methods**: By mapping out the fields and methods of each class, you can plan the implementation details before starting to write code. This helps in avoiding unnecessary refactoring later.

    d. **Understanding Class Interactions**: It allows you to see how classes collaborate and communicate with each other. This is particularly useful in complex systems where understanding the flow of information is crucial.

    e. **Analyzing System Behavior**: UML diagrams can include state diagrams, sequence diagrams, and activity diagrams, which can help in analyzing the behavior of the system under different scenarios.

    f. **Facilitating Communication:** UML diagrams serve as a common language for developers, designers, and stakeholders. They provide a clear and concise way to communicate ideas and concepts about the system.

    g. **Reducing Development Time:** By planning and visualizing the system upfront, you can identify potential issues and design flaws early in the process, which can save time and effort during development.
    h. **Supporting Documentation:** UML diagrams serve as valuable documentation for the system's design. They can be referred to by team members for a clear understanding of the architecture.